Tijdschrift voor Economie en Management
Vol. XLVIII, 1, 2003

Bedrijfsmodellering: de sleutel tot de kwaliteit van informatiesystemen

door M. SNOECK

Monique Snoeck
KULeuven, Departement Toegepaste Economische Wetenschappen, Leuven

ABSTRACT

Het ontwikkelen van kwalitatief hoogstaande informatiesystemen is verre van gemakkelijk. De kwaliteit van informatiesystemen begint bij de kwaliteit van de specificaties. In dit domein is er nog veel ruimte voor verbetering. In dit artikel illustreren we hoe bedrijfsmodellering kan bijdragen tot betere specificaties en tot betere informatiesystemen en tonen we aan wat de extra bijdrage is van de bedrijfsmodelleringsmethode MERODE. We tonen ook aan dat ook wanneer gekozen wordt om software aan te kopen, bedrijfsmodellering een nuttig instrument blijft.

I. PROBLEMATIEK

Het ontwikkelen van softwaresystemen is op vele punten te vergelijken met het bouwen van een huis. In beide gevallen doorloopt het bouwproject een aantal stadia:

- het verzamelen van de wensen van de bouwheer en de gebruikers,
- het opstellen van een plan,
- het opstellen van technische lastenboeken,
- het uitvoeren van de bouwwerkzaamheden.

Na kennis genomen te hebben van de wensen van de bouwheer, zal de architect een plan opstellen. Een eerste punt van vergelijking is dat hoewel het bouwplan gebruik maakt van specifieke symbolen voor de voorstelling van verschillende bouwelementen, het plan voor een leek zeer verstaanbaar is: ook een niet-architect kan zich bij het lezen van een plan een mentaal model voorstellen van het toekomstige huis. Het plan is dus een uitstekend communicatie-instrument voor de architect en de bouwheer. Wie een huis laat bouwen, zal niet nalaten het plan te valideren vóór de eigenlijke bouwfase begint. Het is ook pas wanneer het plan volledig op punt staat, dit wil zeggen, tegemoet komt aan de verwachtingen van de eindgebruiker, dat met de eigenlijke bouwfase begonnen wordt.

Een tweede belangrijk element in de vergelijking met de bouw van informatiesystemen is het feit dan een plan opgebouwd wordt uit een aantal sub-plannen die elk een deelaspect van het gebouw beschrijven: een grondplan per verdieping, gevelschetsen, een volumeschets en eventueel een maquette. Belangrijk is hierbij dat hoewel al deze subplannen apart bekeken kunnen worden, een goede architect er steeds over zal waken dat zij toch steeds in overeenstemming zijn met elkaar. Het verplaatsen van een bouwelement op één plan, zal zonodig leiden tot de aanpassing van de andere plannen.

Indien we nu de vergelijking maken met de bouw van informatiesystemen, merken we dat deze twee elementen niet terug te vinden zijn.

A. *Het plan is een basisinstrument voor de communicatie met de bouwheer en/of eindgebruiker*

De plannen van een informatiesysteem zijn veelal te technisch en moeilijk verstaanbaar voor niet-informatici. Slechts weinig bedrijfsmensen

kunnen zich een mentaal model vormen van het toekomstige informatiesysteem bij het lezen van database-schema's, dataflowdiagrammen, sequence charts,... en dergelijke. Bijgevolg is de communicatie tussen de verantwoordelijken voor de bedrijfsvoering en de ontwikkelaars van informatiesystemen in vele organisaties moeizaam en staan zij aan weerszijden van wat een communicatie-equivalent van de vroegere Berlijnse muur genoemd kan worden. Hoewel beide partijen in eenzelfde bedrijfsomgeving leven, hebben ze meestal te weinig inzicht in elkaars activiteiten en blijft de communicatie beperkt tot het "over de muur gooien" van wensen, specificaties en uiteindelijk ook software.

B. *Het plan bestaat uit een aantal deelplannen die onderling consistent zijn*

Ook bij informatiesystemen worden plannen opgesteld volgens verschillende invalshoeken. De standaard modelleringstaal UML ("Unified Modelling Language" die de te gebruiken symboliek standaardiseert) definieert meer dan negen verschillende invalshoeken en bijhorende schematechnieken om een systeem te beschrijven [6]. Technieken voor het bewaken van de consistentie tussen deze verschillende plannen worden echter niet aangereikt. Bij het plan van een informatiesysteem is het dus best mogelijk dat de verschillende subplannen contradictorisch zijn, zelfs voor belangrijke aspecten. Door de grote complexiteit van informatiesystemen is het bovendien quasi onmogelijk de consistentie handmatig te verifiëren. Inconsistenties worden meestal pas ontdekt tijdens de bouwfase. De kosten van foutencorrectie zijn in deze fase echter veel hoger dan in de specificatiefase en dus is het van groot belang dergelijke fouten en eventuele onvolledigheden reeds in de specificaties op te sporen.

Een groot verschilpunt tussen het bouwen van informatiesystemen en het bouwen van huizen is de frequentie van veranderingen en aanpassingen. Terwijl bij een huis de frequentie van verbouwing redelijk laag ligt, is de behoefte aan flexibiliteit en aanpasbaarheid van informatiesystemen veel groter. Bij het ontwikkelen van informatiesystemen is het dus van zeer groot belang de architectuur zo te ontwerpen dat het systeem gemakkelijk kan aangepast worden aan nieuwe en veranderende eisen van gebruikers en omgeving. Een belangrijk element hierin is de traceerbaarheid van specificaties: indien men op een gemakkelijke manier kan terugvinden waar een bepaalde bedrijfsregel

werd geïmplementeerd, wordt het gemakkelijker de implementatie aan te passen indien deze bedrijfsregel verandert of verdwijnt.

Het onderzoek rond software-engineering in de vakgroep beleidsinformatica concentreert zich rond de algemene problematiek van kwalitatieve systeemontwikkelingen en systeemarchitecturen die dit ondersteunen. Daarbij gaat er vooral aandacht naar specificaties als communicatie-instrument, de consistentie en kwaliteitscontrole van specificaties en flexibele architecturen.

Een van de resultaten van dit onderzoek is de ontwikkeling van de objectgerichte bedrijfsmodelleringsmethode MERODE[1]. Bedrijfsmodellering biedt een deel van het antwoord op bovenvermelde problemen:

- het is een instrument voor verbeterde communicatie met de eindgebruiker en/of opdrachtgever,
- het geeft aanleiding tot een gelaagde structuur voor informatiesystemen, wat leidt tot een verbeterde flexibiliteit en verbeterde onderhoudbaarheid,
- het expliciet behouden van het bedrijfsmodel in een geïmplementeerd systeem zorgt voor de traceerbaarheid van specificaties en leidt aldus tot meer flexibiliteit en onderhoudbaarheid.

MERODE biedt daarbovenop een aantal bijkomende voordelen door het gebruik van een aangepast concept voor object-interactie (met name het concept van bedrijfsgebeurtenissen) en door een verbeterde kwaliteitscontrole door het formeel definiëren van de semantiek achter de symboliek van de verschillende modelleringstechnieken. Tenslotte wordt de gelaagde structuur verder uitgebouwd door het afzonderen van de workflow-aspecten in een workflow-laag.

In de volgende paragrafen lichten we de basisprincipes van bedrijfsmodellering en de bijzondere aspecten van MERODE toe. In de laatste paragraaf tenslotte, bekijken we de richtingen voor verder onderzoek.

II. WAT IS BEDRIJFSMODELLERING?

A. *Een illustratie*

Het principe en het nut van bedrijfsmodellering illustreren we aan de hand van het volgende verhaal dat we ontlenen aan M. Jackson [0].

Een ondernemer woonde naast een meer en besloot haar brood te verdienen door bootjes te verhuren om op het meer te varen. Na een paar seizoenen draaide haar bedrijf heel goed en besloot zij uit te breiden. Om deze uitbreiding voor te bereiden wou zij echter meer informatie over haar bedrijf. Zij ging naar een softwarehuis en vroeg het een informatiesysteem te bouwen dat haar dagelijks de gemiddelde duur van een boottocht zou geven. De analist dacht lang na en schreef toen de volgende specificatie op:

gemiddelde duur van een tocht =
$(\Sigma_{i=1..N}$ (aankomsttijd$_1$ – vertrektijd$_i$))/ N

Hierbij is N het aantal tochten van die dag en loopt i van 1 tot N. Door voor elke tocht de vertrektijd van de aankomsttijd af te trekken, verkrijgen we de duur van de tocht. De gemiddelde duur is de som van alle duurtijden van alle tochten van die dag, gedeeld door het aantal gemaakte tochten.

De programmeur bekeek deze specificatie en bedacht dat dit wel vrij moeilijk te implementeren was: aankomst- en vertrektijden zouden door elkaar geregistreerd worden en die zou hij dan per overeenkomende paren moeten bijeenzetten. Hij was echter knap in wiskunde en bedacht dat als je de aankomsttijden bij elkaar optelt, de vertrektijden bij elkaar optelt en het verschil berekent, dat je dan eveneens de totale duurtijd verkrijgt, zoals uit onderstaand voorbeeld blijkt.

Tocht	vertrektijd	aankomsttijd	duur
1	10	60	50
2	20	40	20
3	25	45	20
Totaal	55	145	90

De gegeven formule kon dus getransformeerd worden tot:

gemiddelde duur van een tocht =
$(\Sigma_{i=1..N}$ (aankomsttijd$_i$) – $\Sigma_{i=1..N}$(vertrektijd$_i$))/N

Deze formule was veel gemakkelijker te implementeren: gewoon alle aankomsttijden en alle vertrektijden optellen en het verschil van deze twee getallen delen door N. Zo gezegd, zo gedaan en de klant was heel tevreden met het resultaat.

Tevreden gebruikers willen meer. Na een tijdje stapte de ondernemer naar het softwarehuis en vroeg om een kleine uitbreiding aan het systeem te bouwen: ze wou graag de duurtijd kennen van de langste boottocht.

De specificatie was gemakkelijk op te stellen. Maar de programmeur was minder opgetogen. Dit was helemaal geen "kleine" uitbreiding, maar vergde het compleet hermaken van het informatiesysteem. Het zou dus veel geld kosten en lang duren. De klant begreep niet dat zo'n eenvoudige vraag zo moeilijk op te lossen zou zijn. Het softwarehuis had er een ontevreden klant bij.

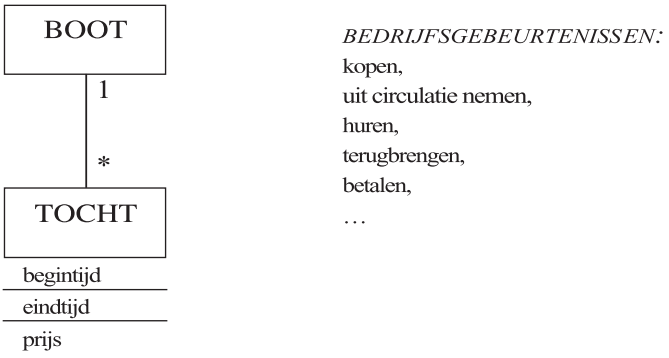
Wat ging er mis? De transformatie die werd uitgevoerd in de ontwerpfase verwijderde de expliciete representatie van de boottocht uit het systeem. In de reële wereld van de ondernemer draait echter alles rond boottochten. Vandaar dat wat een kleine verandering is voor de klant wel een grote verandering is in het informatiesysteem. Had de programmeur de eerste specificatie ongewijzigd geïmplementeerd, dan was het begrip boottocht aanwezig geweest in het systeem en ware het een koud kunstje geweest de duurtijd van de langste boottocht te vinden. Met dit voorbeeld willen we duidelijk maken dat bij het ontwikkelen van informatiesystemen het belangrijk is inzicht te hebben in de reële wereld waarvoor het informatiesysteem wordt gebouwd. Bovendien moet dit begrip van de reële wereld behouden blijven in alle modellen in alle fasen van systeemontwikkeling, tot en met in het implementatiemodel.

B. *Modelgedreven systeemontwikkeling*

Omdat het zo belangrijk is goed inzicht te verwerven in de bedrijfsaspecten van de onderneming waarvoor men een informatiesysteem ontwikkelt, maakt MERODE bij het ontwikkelen van informatiesystemen expliciet een onderscheid tussen het "bedrijfsmodel" en het "functionaliteitsmodel". Het bedrijfsmodel bevat alle specificaties die te maken hebben met de reële wereld van het bedrijf en wordt gespecificeerd in termen van objecten, relaties tussen objecten, gebeurtenissen en bedrijfsregels. Het functionaliteitsmodel bevat de specificaties die gerelateerd zijn aan het informatiesysteem. Het wordt gedefinieerd in termen van invoer- en uitvoerfuncties: dit zijn informatiediensten die respectievelijk instaan voor het invoeren van gegevens in het bedrijfsmodel of het opvragen van informatie uit het bedrijfsmodel.

FIGUUR 1

Bedrijfsmodel voor het bootverhuurbedrijf



Figuur 1 geeft een voorbeeld – in UML notatie – van wat een bedrijfsmodel voor het bootverhuurbedrijf zou kunnen inhouden. Links staat een object-relatiediagramma dat de bedrijfsobjecten BOOT en TOCHT voorstelt. De relatie tussen deze objecten stelt dat elke tocht door 1 boot gemaakt wordt en dat men met een boot meerdere (*) tochten kan maken. Per tocht houden we waarden bij voor de begintijd, de eindtijd en de gevraagde prijs. Dergelijk object-relatiediagramma beschrijft de statische structuur van het domein in termen van objecten, relaties tussen objecten en attributen. De dynamische aspecten kan men onder meer beschrijven door middel van bedrijfsgebeurtenissen. Figuur 1 identificeert voor het bootverhuringsbedrijf onder meer de gebeurtenissen: *kopen, uit circulatie nemen, huren* en *terugbrengen* van een boot en het *betalen* van een boottocht.

Het bedrijfsmodel bevat enkel concepten uit de reële wereld en is volledig onafhankelijk van informatica-technische aspecten. Daarom is het een goed instrument voor communicatie met de eindgebruiker. Zelfs indien het bedrijfsmodel niet gebruikt wordt voor het ontwikkelen van een informatiesysteem, is het een uitstekend instrument in het ontwikkelen van een gemeenschappelijk taal en het verzamelen van business rules binnen het bedrijf. Het gebeurt immers vaak dat verschillende afdelingen en/of mensen een verschillende betekenis hechten aan een aantal basisbegrippen. Denk bijvoorbeeld aan vragen zoals:

– Vanaf wanneer wordt een persoon binnen het bedrijf als klant beschouwd?

– Kan een persoon klant zijn indien hij/zij nog nooit een bestelling geplaatst heeft?
– Vanaf wanneer beschouwt men een idee voor een nieuw product als feitelijk nieuw product met alles wat erbij hoort?
– Kan een levering op verschillende orders betrekking hebben?
– Moet een order in één keer geleverd worden of kunnen er deelleveringen zijn?

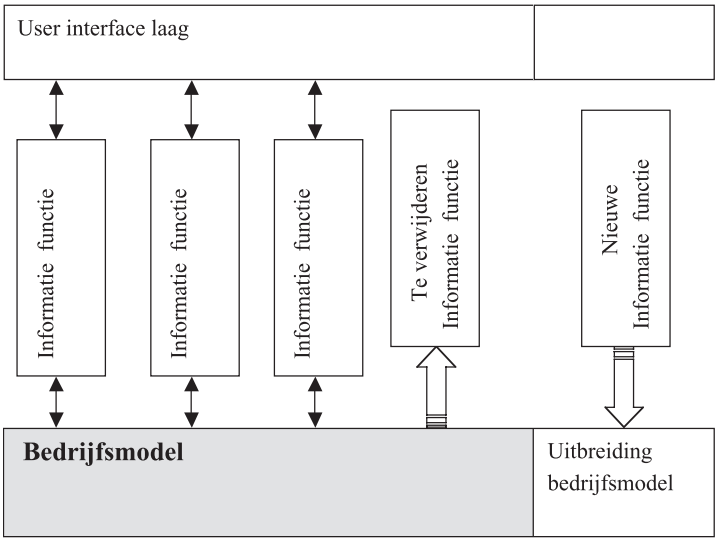
Eens men een bedrijfsmodel heeft gebouwd, kan men het informatiesysteem specificeren als een aantal informatiefuncties rond dit bedrijfsmodel. Voor het gegeven voorbeeld zijn mogelijke informatiefuncties: het registreren van een nieuwe boot, het registreren van het begin, het einde, de betaling van een boottocht, het opvragen van de gemiddelde duur van boottochten per dag, per maand, per seizoen, …, het opvragen van de gemiddelde opbrengst per boot per dag, per week, per jaar, … en zo verder.

C. *Gelaagde systeemarchitectuur*

Als gevolg van deze modelgedreven benadering, krijgen systemen een gelaagde architectuur. Het bedrijfsmodel vormt de onderste laag. Daarboven komt de functionaliteitslaag. Zoals verderop in dit artikel wordt uiteengezet, kunnen er nog bijkomende lagen gedefinieerd worden en kan elke laag nog eens verder opgesplitst worden. Het algemene principe is dat onderste lagen niets afweten van de lagen die erboven liggen. Objecten in een bepaalde laag maken alleen gebruik van objecten in dezelfde laag of van objecten in de onderliggende lagen. Als men er bovendien voor zorgt dat de onderste laag de meest stabiele is, verhoogt men de stabiliteit van het informatiesysteem. Concreet gezien bestaat een MERODE-systeem uit minstens 3 lagen: de bedrijfslaag die de realisatie is van het bedrijfsmodel, de functionaliteitslaag en de user-interface laag. Als algemene regel geldt dat informatiefuncties onafhankelijk van elkaar zijn: de informatiedoorstroming van een functie naar een andere gaat via het bedrijfsmodel. De user interface lijmt de functies aan elkaar tot een ergonomisch geheel. Als gevolg hiervan kunnen functies uit het systeem worden weggehaald en aan het systeem toegevoegd zonder dat de onderste laag hierdoor moet aangepast worden. Ook een uitbreiding van het bedrijfsmodel heeft geen invloed op de bestaande functionaliteit, maar laat wel toe om nieuwe functies te implementeren. Dit wordt schematisch weergegeven in Figuur 2.

FIGUUR 2

Systeemarchitectuur volgens MERODE



De flexibiliteit van deze architectuur is veel hoger dan die van een monolitische applicatie. Wijzigingen in de bovenste lagen hebben immers geen invloed op de onderliggende lagen. De structuur is zo opgezet dat de elementen die de hoogste frequentie van verandering kennen in de bovenste lagen zitten. Het meest stabiele deel van het systeem zit in de onderste laag.

III. SPECIFIEKE EIGENSCHAPPEN VAN MERODE

A. *MERODE biedt een aangepast concept aan voor objectinteractie*

Objectgerichtheid is ontstaan in de context van programmeertalen. Later is men dezelfde principes gaan gebruiken bij het ontwerpen van systemen en nog later is men deze principes gaan toepassen op analyseniveau. Telkens heeft men de basisprincipes van objectgerichtheid zoals ze ontstaan zijn voor programmeertalen, overgenomen zonder ze aan te passen aan het nieuwe abstractieniveau. Nochtans valt voor sommige principes te argumenteren dat ze een andere invulling moeten

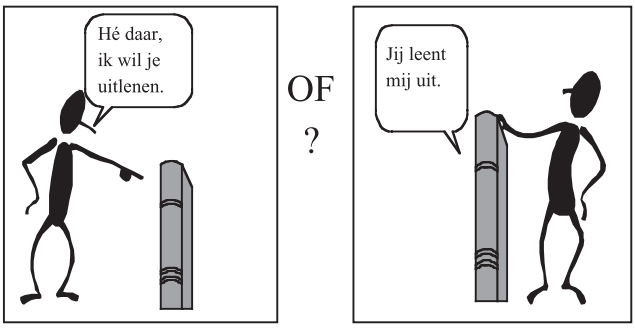
krijgen op analyseniveau. In dit artikel illustreren we dit voor object-interactie[2].

Laten we uitgaan van het bibliotheek-domein, waar leden boeken kunnen ontlenen. Omdat er van een boek meerdere exemplaren kunnen bestaan, gebruiken we in wat volgt de naam "copie" voor het aanduiden van een exemplaar van een boek. Een eerste versie van een bedrijfsmodel voor de bibliotheek bevat de objecttypes COPIE en LID en het gebeurtenistype *uitlenen.* Wanneer een uitleengebeurtenis plaats heeft, zijn zowel een lid als een copie (een exemplaar van een boek) betrokken bij deze gebeurtenis en ontstaat er dus interactie tussen deze twee objecten. In de klassieke objectgerichte benadering wordt object-interactie gemodelleerd door middel van berichten die door het ene object naar het andere worden gestuurd. Concreet betekent dit voor het bibliotheekdomein dat ofwel het boek het bericht *uitlenen* naar het lid verstuurt ofwel andersom (zie Figuur 3). De eventuele keuze tussen deze twee scenario's is echter niet relevant op bedrijfsmodelleringsniveau: het voordeel van het ene scenario ten opzichte van het andere heeft alles te maken met ontwerp- of implementatie-aspecten en niets met bedrijfsregels.

Stel nu dat er drie objecten deelnemen aan één gebeurtenis, namelijk COPIE, LID en UITLENING (dit laatst object ontstaat wanneer een exemplaar ontleend wordt en eindigt als dit teruggebracht wordt). In dit geval zijn er al 9 mogelijke scenario's (zie Figuur 4). COPIE, LID en UITLENING zijn alledrie betrokken bij de gebeurtenis *uitlenen*: de uitlening wordt gecreëerd, de toestand van de copie moet aangepast worden en voor het lid moet gecontroleerd worden of hij/zij nog niet teveel boeken in leen heeft.

FIGUUR 3

Object-interactie in de bibliotheek



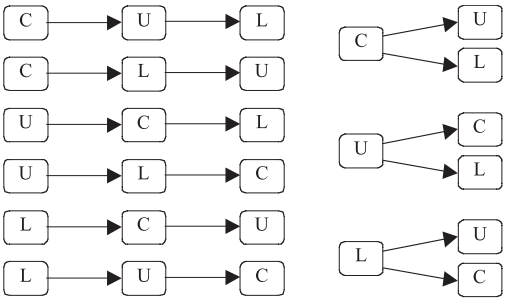
Stel nu dat er in dit model nog een vierde object op de hoogte moet gebracht worden van de gebeurtenis uitlenen, bv. een object TITEL dat het abstracte begrip van een boek voorstelt waarvan er verschillende exemplaren in de bibliotheek kunnen beschikbaar zijn. Wanneer er vier objecten moeten interageren bij het optreden van een gebeurtenis, explodeert het aantal mogelijkheden. Figuur 5 geeft een *niet-exhaustieve* lijst van mogelijkheden wanneer een objecttype TITEL wordt toegevoegd aan het bibliotheekvoorbeeld. De ordegrootte van de snelheid waarmee het aantal mogelijke scenario's toeneemt is groter dan N!, waarbij N het aantal deelnemende objecten voorstelt.

Het is duidelijk dat het uitwisselen van berichten (of "message passing") een aantal nadelen heeft, althans voor het opstellen van een model van de reële wereld[3]. Het meest belangrijke nadeel is dat *message passing altijd binair is en unidirectioneel.* Message passing laat uitsluitend een binaire en unidirectionele vorm van interactie toe: een bericht wordt verstuurd van het eerste object naar het tweede object. In de reële wereld is interactie echter niet altijd binair en unidirectioneel. Een bedrijfsmodel zou een weergave moeten zijn van de reële wereld en moet dus over een interactieconcept beschikken dat algemener is dan message passing.

Omwille van dit en nog andere nadelen maar vooral omdat het kiezen van een message-passing scenario gebeurt op basis van overwegingen die niet mogen meespelen op conceptueel analyse-niveau, gebruikt MERODE het principe van "communicatie door synchrone deelname aan gezamenlijke gebeurtenissen". Voor elke bedrijfsgebeurtenis wordt nagegaan welke bedrijfsobjecten erbij betrokken zijn en wat het effect is van de gebeurtenis op deze objecten (creatie,

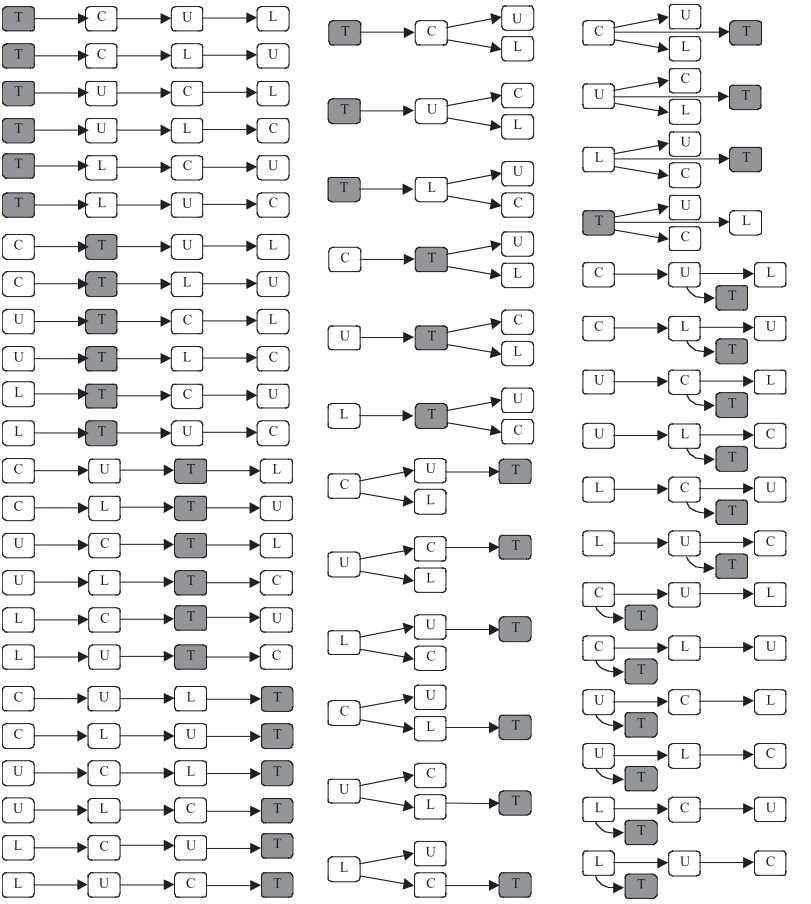
FIGUUR 4

Mogelijke scenario's wanneer 3 objecten met elkaar moeten interageren bij het optreden van een gebeurtenis. (C = copie, U = uitlening, L = lid)



FIGUUR 5

Niet exhaustieve lijst van mogelijke scenario's wanneer 4 objecten moeten synchroniseren bij het optreden van een gebeurtenis (C = copie, U = uitlening, L = lid, T = titel)



wijziging of beëindiging). Dit wordt weergegeven in een object-eventtabel. Deze tabel heeft een rij per gebeurtenistype en een kolom per bedrijfsobjecttype. Een cel wordt gemarkeerd met een 'C' een 'M' of een 'E' als een gebeurtenis betrekking heeft op het object en dit respectievelijk creëert (Creator), wijzigt (Modifier) of beëindigt (Ending-event).

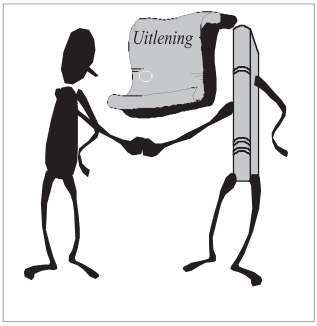
Het markeren van een cel betekent dat dit bedrijfsobjecttype voorzien wordt van een procedure die het effect van de gebeurtenis op

objecten van dit type weergeeft. Bij het optreden van de gebeurtenis, zullen alle procedures die voor deze gebeurtenis gedefinieerd zijn, simultaan uitgevoerd worden. Dit wordt schematisch weergeven in Figuur 6 voor het bibliotheekvoorbeeld.

Tabel 1 geeft de volledige object-eventtabel voor het bibliotheekvoorbeeld, waarbij we uitgaan van de vier objecttypen TITEL, COPIE, LID en UITLENING. De object-eventtabel heeft één kolom per objecttype en één rij per gebeurtenistype dat geïdentificeerd werd binnen dit domein. In de tabel wordt door middel van een C, M of E aangegeven of een gebeurtenis objecten van dit type Creëert, wijzigt (Modify) of beëindigt (End). Het bouwen van een object-eventtabel gebeurt aan de hand van een aantal strikte consistentieregels. Zo moet er onder andere per objecttype minstens één gebeurtenis zijn die objecten van dit type creëert en minstens één gebeurtenis die objecten van dit type beëindigt. Andere regels bewaken bijvoorbeeld de consistentie van de object-eventtabel met het object-relatiediagramma (zie volgende paragraaf). Een uiteenzetting van al deze regels kan gevonden worden in [10, 11, 12]. Tabel 1 toont een maximaal aantal object-event participaties: ze inventariseert alle mogelijke plaatsen voor het specificeren van bedrijfsregels. Indien achteraf blijkt dat een aantal gemarkeerde cellen toch geen aanleiding hebben gegeven tot bedrijfsregels, kunnen de nodige optimisaties doorgevoerd worden bij implementatie. Het eerste belangrijke voordeel van deze manier van werken is dat *gebeurtenissen als zelfstandige begrippen, los van bedrijfsobjecten, kunnen beschouwd worden.*

FIGUUR 6

Partiële object-eventtabel voor het bibliotheekvoorbeeld



	COPIE	LID	UITLENING
uitlenen	M	M	C
hernieuwen	M	M	M
terugbrengen	M	M	E

Simultaan uitvoeren van
LID.uitlenen
COPIE. uitlenen
UITLENING.uitlenen

TABEL 1
Object-Eventtabel voor het bibliotheekvoorbeeld

	lid	titel	copie	uitlening
creëren_lid	C			
wijzigen_lid	M			
einde_lid	E			
creëren_titel		C		
wijzigen_titel		M		
einde_titel		E		
creëren_copie		M	C	
classificeren		M	M	
uitlenen	M	M	M	C
hernieuwen	M	M	M	M
terugbrengen	M	M	M	E
verliezen	M	M	E	E
einde_copie		M	E	

Het inventariseren van bedrijfsgebeurtenissen kan gebeuren door te kijken naar creatie-, wijzigings- en beëindigingsgebeurtenissen van bedrijfsobjecten (Hoe ontstaat een lening? Hoe wordt een lening beëindigd?...) maar bijvoorbeeld ook door te gaan kijken naar de bedrijfsprocessen. Wanneer men bijvoorbeeld kijkt naar een verkoopsproces vindt men gemakkelijk een hele reeks gebeurtenissen zoals het creëren van een klant, het creëren van een order, het creëren van een orderregel, het ondertekenen van een order, het leveren, het factureren, het ontvangen van betalingen, … en zo verder. Voor elk van deze gebeurtenissen kan men dan nagaan welke objecten erbij betrokken zijn. Dit leidt eventueel tot de ontdekking van ontbrekende bedrijfsobjecten. Ook databeperkingen en beperkingen op de volgorde waarin gebeurtenissen ten opzichte van elkaar mogen voorkomen, kunnen, althans initieel, gebeurtenis per gebeurtenis onderzocht worden.

Een tweede belangrijk voordeel is dat een bedrijfsgebeurtenis een uniek aanspreekpunt wordt voor het manipuleren van bedrijfsobjecten.

In plaats van berichten naar bedrijfsobjecten te sturen, kan een informatieservice nu de uitvoering van een gebeurtenis activeren en zal het overeenkomstige gebeurtenis-object zorgen voor de dispatching naar alle deelnemende bedrijfsobjecten. Dit is vooral interessant wanneer dezelfde functionaliteit wordt aangeboden onder verschillende vormen. Denk bijvoorbeeld aan het afhalen van geld dat op meerdere manieren kan gebeuren: aan het loket, via Bancontact, via de bank-eigen ATM automaten of via het herladen van de Proton-kaart, eventueel zelfs over het Internet. Voor elk van deze diensten bestaat er een eigen geheel aan software-functies. In een event-loze aanpak zal elk van deze diensten zijn eigen message-passing schema moeten ontwikkelen. In een eventgerichte benadering daarentegen, kunnen al deze diensten gebruik maken van de bedrijfsgebeurtenis "afhalen" en zal de gebeurtenis zelf instaan voor het bijwerken van alle relevante bedrijfsobjecten door middel van zijn eigen dispatching en coördinatiemechanisme.

Over de voor- en nadelen van message passing versus de object-eventtabel vindt u een uitgebreidere discussie in [8].

B. *MERODE biedt een betere kwaliteitscontrole door formele definitie van de semantiek van de notatietechnieken en uitgebreide controleregels*

Een van de grootste problemen met de huidige technieken binnen objectgerichte systeemanalyse is de onvolledige en/of niet precieze definitie van de notaties.

Neem als voorbeeld het object-relatieschema in Figuur 7. Het schema bevat de objecttypen KLANT, PROJECT en WERKNEMER. De lijnen die het objecttype KLANT met het objecttype PROJECT en het objecttype PROJECT met het objecttype WERKNEMER verbinden, stellen respectievelijk de relaties "bestelt" en "werkt-voor" voor. Een klant bestelt 0, 1 of meerdere projecten (wat wordt aangegeven door het interval [0..*] aan de kant van het objecttype PROJECT) en een project is besteld door exact 1 klant (wat wordt aangegeven door het interval [1..1] aan de kant van het objecttype KLANT. Analoog werken op een project 0, 1 of meerdere werknemers en werkt een werknemer voor 1 project. De UML notatie voor beide relaties is identiek, terwijl beide relaties toch in essentie verschillend zijn. In het bijzonder is de relatie "bestelt" een bestaansafhankelijke relatie terwijl de relatie "werkt-voor" dat niet is. Een project kan niet bestaan tenzij het besteld werd door een klant. Bovendien is het voor eens en voor altijd besteld door diezelfde klant:

FIGUUR 7

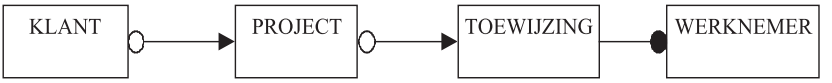
Projectmanagement

het is onmogelijk dat een project dat vandaag door klant A besteld is, morgen door klant B besteld zou zijn. De relatie "bestelt" tussen klant en project is m.a.w. onveranderlijk. De relatie tussen project en werknemer is dat niet: een werknemer kan vandaag op project 1 werken en morgen op project 2. Bijgevolg is een werknemer ook niet bestaansafhankelijk van een project, noch andersom. Het schema in Figuur 7 geeft dus een onvolledige definitie van de structuur van het domein. De voorgaande discussie is ook sterk gerelateerd met het tijdsperspectief waarin de cardinaliteiten moeten geïnterpreteerd worden: een werknemer werkt aan één project op één ogenblik in de tijd, doch kan aan meerdere projecten werken door de tijd heen. Een project is besteld door één klant op één ogenblik in de tijd, maar is dat ook door de tijd heen.

Om ambiguïteiten met de klassiek genoteerde relaties te voorkomen, heeft MERODE een eigen notatie voor het weergeven van relaties. In MERODE gebruiken we een bestaansafhankelijkheidsschema ter vervanging van het klassiek object-relatiediagramma [10, 11, 12]. Hoewel het niet vanzelfsprekend lijkt dat alle relaties tussen objecten kunnen gemodelleerd worden door middel van bestaansafhankelijke relaties is dat toch vrij eenvoudig. Het basisprincipe is dat elke relatie ofwel bestaansafhankelijkheid uitdrukt ofwel niet. In het eerste geval blijft de relatie zoals ze is. In het tweede geval wordt de relatie omgezet naar een nieuw objecttype dat bestaansafhankelijk is van de objecttypen die deelnemen aan de relatie. In het projectmanagement-voorbeeld is de "werkt-voor" relatie geen bestaansafhankelijkheidsrelatie: een project hangt voor zijn bestaan immers niet af van een werknemer, noch omgekeerd. Daarom wordt ze omgezet naar een objecttype TOEWIJZING. Dit objecttype is wel bestaansafhankelijk van de objecten die door de relatie "werkt-voor" gerelateerd werden, m.n. van PROJECT en WERKNEMER: een toewijzing kan niet bestaan tenzij de toegewezen werknemer en het project bestaan. Een toewijzingsobject vertegenwoordigt de periode dat een werknemer toegewezen is aan een project. Het overeenkomstige bestaansafhankelijkheidschema wordt gegeven in Figuur 8. Hierin duiden een witte, respectievelijk zwarte bol aan dat het hebben van een bestaansafhankelijk object

FIGUUR 8

Bestaansafhankelijkheidsschema voor het project management voorbeeld



optioneel, respectievelijk verplicht is en geven het pijlvormig of rechte einde respectievelijk weer dat er meerdere, respectievelijk maximaal 1 bestaansafhankelijk object kan zijn op één ogenblik in de tijd. We zien in deze figuur dus dat een klant meerdere projecten kan hebben maar dat het hebben van projecten optioneel is; dat aan een project via TOEWIJZING meerdere werknemers kunnen toegewezen worden maar ook geen en dat een werknemer op elk ogenblik verplicht aan exact één project toegewezen is. Een toewijzing kan niet bestaan tenzij de werknemer en het project bestaan en een project kan slechts bestaan indien de bestellende klant bestaat.

Omdat zo'n relatie-objecttype (hier: toewijzing) een eigen levenscyclus heeft en eigen bedrijfsgebeurtenissen voor de creatie en de beëindiging van objecten, gaat het hier niet louter om een kwestie van notaties. In het gegeven voorbeeld vereist het bestaan van het objecttype TOEWIJZING het definiëren van op zijn minst 2 gebeurtenissen: *toewijzen* voor het toewijzen van een werknemer aan een project en *verwijderen* voor het wegnemen van een werknemer van een project [10, 11, 12]. Het nieuwe objecttype TOEWIJZING kan ook beschouwd worden als een soort *contract* tussen WERKNEMER en PROJECT waarin alle condities worden vastgelegd: hoe het begint, hoe het eindigt, wat er tussenin kan gebeuren en alle voorwaarden die daarmee verbonden zijn. Door het expliciet maken van het objecttype TOEWIJZING krijgt men ook extra mogelijkheden op het vlak van informatievergaring en -functionaliteit. Zo kan het model bijvoorbeeld verder uitgebreid worden met de registratie van de tijd die werknemers aan een project besteden. Dit kan gemodelleerd worden met een objecttype REGISTRATIE dat bestaansafhankelijk is van TOEWIJZING.

In MERODE zijn alle modelleringstechnieken formeel gedefinieerd door middel van een procesalgebra. Gebruikers van de methode hoeven de formele definities niet te kennen. Door het formaliseren van de technieken was het echter mogelijk om te definiëren wanneer specificaties consistent zijn en wanneer niet. Hieruit werden een aantal consistentieregels afgeleid. Deze helpen de analist bij het opsporen van fouten of onvolkomenheden in de specificaties.

C. *MERODE streeft een betere integratie na van het informatiesysteem met de organisatie-aspecten van het gehele "werksysteem" waarin het informatiesysteem is ingebed*

Veel bedrijfsinformatiesystemen worden ontwikkeld als ondersteuning of automatisering van taken binnen het bedrijf, al dan niet na het herdenken van bedrijfsprocessen. Het organiseren, modelleren en automatisch opvolgen van taken behoort tot het domein van workflowsystemen. Workflowsystemen en objectgerichte systeemontwikkeling zijn twee onderzoeksdomeinen die zich los van elkaar ontwikkeld hebben. Dit is verwonderlijk omdat men zou kunnen verwachten dat er vanuit werkorganisatorische aspecten heel wat richtlijnen kunnen komen voor de ontwikkeling van informatiesystemen die deze werkorganisatie moeten ondersteunen. Het gebrek aan aandacht voor de organisatie-aspecten binnen de huidige objectgerichte systeemontwikkelingspraktijk[4] maakt systemen weinig flexibel: reorganisatie van het werk vereist meestal een (moeilijke) aanpassing van het systeem omdat de organisatorische aspecten niet afgezonderd werden van andere aspecten van het systeem.

Om organisatie-aspecten op een adequate manier in te brengen in systeemontwikkeling moet rekening gehouden worden met de volgende vereisten [9]:

1. Processen en activiteiten zijn hiërarchisch gestructureerd en moeten ook op deze wijze kunnen geanalyseerd en gedefinieerd worden.
2. De afhankelijkheden tussen taken en activiteiten zijn een cruciaal element in de definitie van de werkorganisatie. Het modelleren van deze afhankelijkheden vormt de kern van workflow modellering.
3. De interactie tussen mensen (human agents), computer agents, de rollen die ze hebben en de toewijzing van taken aan rollen is eveneens een belangrijk element in workflow modellering.
4. De specificatie van bedrijfsprocessen moet een persistent gegeven zijn dat de kern vormt van een informatiesysteem (m.n. een workflow engine) dat de uitvoering van taken kan controleren, superviseren en monitoren.

De huidige objectgerichte systeemontwikkelingsmethoden bieden onvoldoende ondersteuning voor een adequate modellering van de organisatieaspecten.

Eerst en vooral is in de objectgerichte wereld het woord "decompositie" synoniem geworden voor "slecht" [13]. Alhoewel functionele

decompositie inderdaad geen goede basis vormt voor systeemarchitectuur, is de notie van decompositie wel nuttig om de hiërarchisch gestructureerde processen en activiteiten te kunnen beschrijven (cfr. vereiste 1). Men heeft de filosofie van functionele decompositie ten dele heringevoerd met de techniek van Use Cases. Use Cases beschrijven de functionele vereisten van een systeem door het identificeren van gebruikers en van gebruiksscenario's. Dit laat toe om de interactie tussen gebruiker en systeem te modelleren. Hiërarchische decompositie is ten dele mogelijk met behulp van de constructies "include" en "extends". Use Cases hebben als eerste doel de functionele vereisten ten aanzien van het systeem vast te leggen. Ze zijn helemaal niet bedoeld om de organisatie van het werk te modelleren: ze worden niet gebruikt om taken toe te wijzen aan agents noch om de afhankelijkheden tussen taken in kaart te brengen (vereisten 2 en 3). Zelfs indien men Use Cases daarvoor zou gebruiken, zijn ze niet bedoeld om te implementeren als workflow engine (vereiste 4).

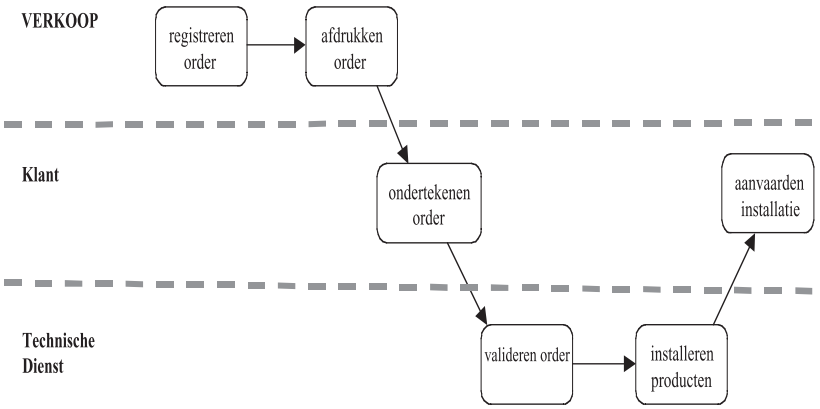
Omwille van hun gelijkenis met Petri-Netten zijn de Activity Diagrams veel beter geschikt voor het modelleren van de organisatieaspecten. Maar ook Activity Diagrams zijn niet bedoeld om te dienen als specificatie voor een Workflow Engine.

Tenslotte kunnen er heel wat businessprocess aspecten verborgen zitten in andere schematechnieken zoals in de toestandsdiagrammen en de object-relatiediagrammen. In dit geval zijn de businessprocess aspecten moeilijk te herkennen en leiden ze tot weinig flexibele systemen. Neem bijvoorbeeld een bedrijf dat producten verkoopt die bij de klant geïnstalleerd moeten worden. Het hele verkoopsproces bestaat uit een aantal stappen waarvan sommige in bepaalde volgorde moeten doorlopen worden. Figuur 9 geeft een mogelijke organisatie van het verkoopsproces weer: het order wordt eerst geregistreerd (d.i. ingegeven en aangepast tot de klant tevreden is); het order wordt dan afgedrukt en ondertekend door de klant. Na ondertekening wordt het order aanvaard door de technische dienst die het opneemt in zijn planning; tenslotte worden de producten geïnstalleerd bij de klant. Het order wordt als afgesloten beschouwd wanneer de klant de installatie van de producten heeft aanvaard.

Figuur 10 en Figuur 11 geven alternatieven voor de organisatie van het verkoopsproces. In Figuur 10 kan de technische dienst het order al opnemen in de planning nog vóór het officieel ondertekend is door de klant. Bovendien moet het order door de technische dienst gevalideerd worden vóór het kan ondertekend worden door de klant. In

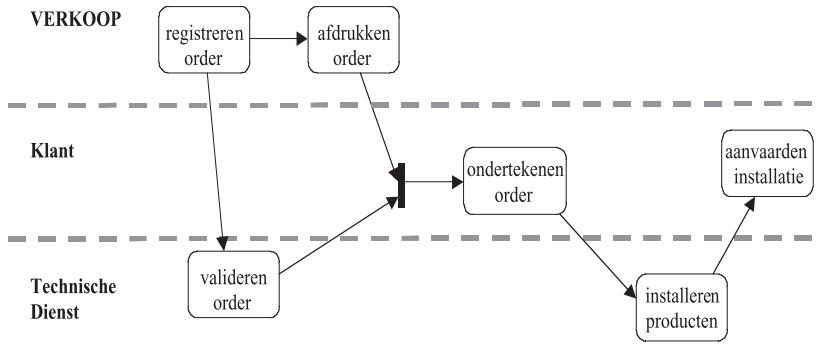
FIGUUR 9

Mogelijke organisatie van het verkoopsproces



FIGUUR 10

Alternatieve organisatie van het verkoopsproces

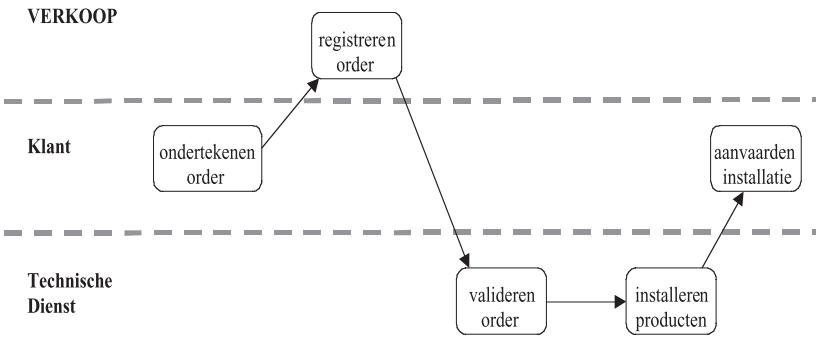


Figuur 11 wordt het order pas ingegeven als het definitief op punt staat.

Om een vlotte aanpassing van de bedrijfsprocessen mogelijk te maken is het belangrijk volgordebeperkingen die het gevolg zijn van werkorganisatie *niet* op te nemen in het bedrijfsmodel. Zo creëert men de nodige ruimte om bedrijfsprocessen te reorganiseren zonder het bedrijfsmodel te moeten wijzigen. Indien bijvoorbeeld het bedrijfsobjecttype ORDER een bepaalde volgorde zou opleggen aan de bedrijfsgebeurtenissen *ondertekenen order* en *valideren order*, dan is het veel

FIGUUR 11

Alternatieve organisatie van het verkoopsproces



moeilijker om van de ene procesorganisatie naar de andere over te gaan. Het is beter om dergelijke volgordebeperkingen te lokaliseren in het workflowsysteem in plaats van in het bedrijfsmodel.

De expliciete identificatie van bedrijfsgebeurtenissen in het bedrijfsmodel maakt het bovendien gemakkelijker om een verband te leggen tussen taakdefinities en het bedrijfsmodel: bedrijfsprocessen beschrijven het domein vanuit een dynamisch perspectief en zijn dus gemakkelijker te formuleren in termen van bedrijfsgebeurtenissen dan in termen van objecttypes en relaties tussen objecttypes.

De analyse van het bedrijf kan verder verfijnd worden door het classificeren van taken als manuele taken, interactieve taken of geautomatiseerde taken. Interactieve en geautomatiseerde taken zullen aanleiding geven tot de definitie van diensten in het informatiesysteem. Deze 'information system services' vormen de interface tussen het bedrijfsmodel en het workflow systeem. Bij het uitvoeren van taken maken mensen of computers gebruik van deze diensten voor het initiëren van bedrijfsgebeurtenissen.

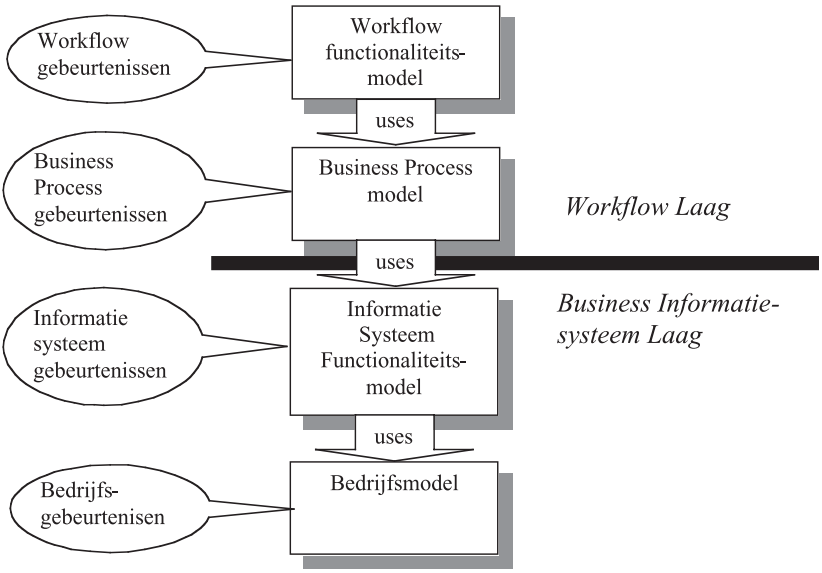
Het afzonderen van de workflow-aspecten geeft aanleiding tot extra lagen in de MERODE-architectuur. Het bedrijfsmodel vormt de onderste en meest stabiele laag. Daarboven komt de informatiesysteem-laag (inclusief user interface-laag) die bestaat uit een aantal onafhankelijke input- en outputdiensten die toelaten om informatie te registeren in het bedrijfsmodel door gebeurtenissen te activeren en om informatie uit het bedrijfsmodel te halen. Daarboven komt dan de business process-laag. Ook in deze laag kan men een onderscheid maken tussen de beschrijving van het business process model

en het workflow-informatiesysteem. Het business process model beschrijft concepten zoals AGENT, WORKFLOW ACTIVEIT, BUSINESS PROCESS, TAAKAFHANKELIJKHEID, TAAKLIJST, en zo verder. Door de objecttypes in het business process model te populeren met concrete activiteiten, business processen, agenten, afhankelijkheden, taaklijsten en dergelijke definieert men een gegeven bedrijfsorganisatie. Dit gepopuleerde business process model fungeert als persistent model van de bedrijfsprocessen in de organisatie zoals beschreven in vereiste 4 hiervoor. Het workflow informatiesysteem voorziet in de diensten die nodig zijn om bijvoorbeeld nieuwe taken toe te voegen, taken te schrappen, statusveranderingen van taken te registreren, taaklijsten aan te maken … en zo verder. Deze vier lagen worden voorgesteld in Figuur 12.

De dynamische aspecten van elke laag worden geactiveerd door verschillende soorten gebeurtenissen. Workflow-systeemgebeurtenissen zijn gebeurtenissen die gerelateerd zijn aan het gebruiken van een workflowsysteem. Het gaat hier om ICT-gebeurtenissen zoals het indrukken van toetsen op muis en/of klavier. Business process gebeurtenissen treden op in de reële wereld en activeren de dynamische

FIGUUR 12

Vier basislagen van de systeemarchitectuur



aspecten van het bedrijfsprocessen-domein. Het gaat hier om gebeurtenissen zoals de creatie van een nieuwe taak, het uitvoeren van een taak, het toewijzen van een taak aan een werknemer. Informatiesysteemgebeurtenissen zijn ICT-gebeurtenissen gerelateerd aan het gebruik van het informatiesysteem. Zij treden op wanneer het informatiesysteem wordt gebruikt bij het uitvoeren van de taken gedefinieerd in het business process model. Tot slot zijn de bedrijfsgebeurtenissen de triggers van het gedrag van de bedrijfsobjecten. Veronderstel bijvoorbeeld dat de installatie van een order wordt toegewezen aan een bepaalde Service Engineer (Business Process aspect). Deze toewijzing gebeurt door het order te registreren in de taaklijst van deze persoon (Workflow aspect). De Service Engineer zal te gepasten tijde de producten uit het order installeren bij de klant (bedrijfsgebeurtenis) en zal de geïnstalleerde producten registreren in het informatiesysteem (informatiesysteem-aspect). Tegelijk kan de taak als "afgewerkt" worden gemarkeerd in het Workflow-systeem.

IV DE ROL VAN BEDRIJFSMODELLERING IN OUTSOURCING-, ERP- EN INTEGRATIE-PROJECTEN

Met het toenemend gebruik van ERP-software neemt het belang van eigen ontwikkelde software meer en meer af. En dus kan men zich terecht de vraag stellen of bedrijfsmodellering een rol te spelen heeft wanneer een bedrijf al haar software aankoopt en niet meer zelf ontwikkelt.

Ook in dergelijke omgevingen heeft bedrijfsmodellering haar nut. In de eerste plaats blijft een bedrijfsmodel een nuttig instrument voor het verzamelen en op punt stellen van de bedrijfsregels. Het bedrijfsmodel definieert immers de belangrijkste concepten binnen het bedrijf en hoe met deze concepten kan worden omgegaan. Het is niet uitzonderlijk te moeten vaststellen dat het concept "product" een verschillende betekenis heeft al naargelang de afdeling die het concept moet definiëren. Bovendien zal het bedrijfsmodel ook vastleggen wat de levensloop van een product in het bedrijf is: vanaf wanneer wordt een product als "bestaand" beschouwd en wordt het officieel geregistreerd. Vanaf wanneer kan een product besteld worden en hoe gebeurt het stopzetten van de verkoop? Het specificeren van bedrijfsregels ten behoeve van een informatiesysteem vergt immers een grote nauwkeurigheid. Computers kunnen niet met ambigue definities om en

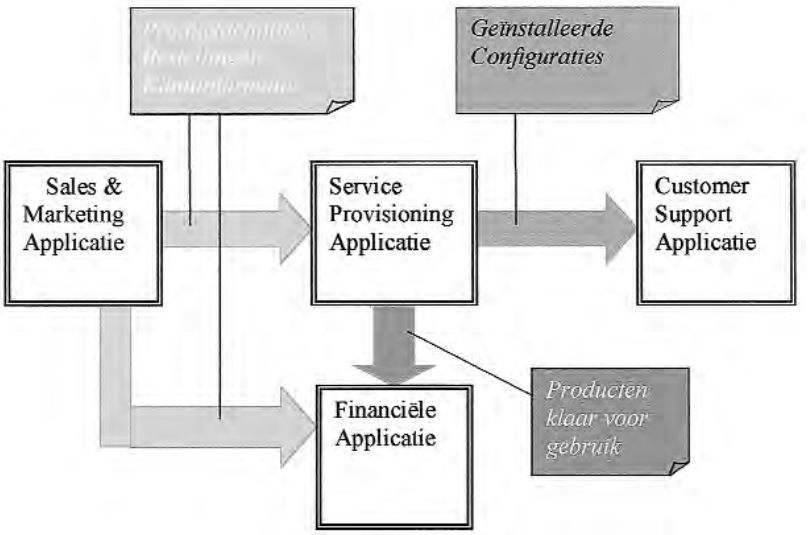
daarom moeten specificaties zeer precies zijn. Als dusdanig is bedrijfsmodellering dus een krachtig instrument voor het ontwikkelen van een bedrijfswijde verzameling van bedrijfsregels, ook indien het bedrijf niet van plan is software zelf te ontwikkelen.

Gezien bedrijfsregels een essentieel onderdeel zijn van een bedrijfsinformatiesysteem, vormen zij de basis waarop een outsourcingproject kan geënt worden. Ook bij de aankoop van softwarepakketten is een bedrijfsmodel een nuttig instrument. Het laat immers toe de bedrijfsregels te toetsen aan de functionaliteiten van het pakket en is aldus een nuttig instrument bij de beoordeling van softwarepakketten en bij de voorbereiding van de parametrisatie van een pakket.

Wanneer verschillende softwarepakketten gecombineerd worden, kan een bedrijfslaag gebruikt worden als integratielaag. De aangekochte pakketten functioneren dan als input en output-services ten opzichte van de bedrijfslaag. Deze aanpak heeft als voordeel dat de informatieflow tussen de verschillende pakketten niet wordt vastgelegd in de implementatie. Figuur 13 geeft een voorbeeld van de klassieke manier van integratie van softwarepakketten. Doordat de verschillende softwarepakketten rechtstreeks informatie uitwisselen, wordt het basisbedrijfsproces hard gecodeerd in de geïntegreerde

FIGUUR 13

Voorbeeld van applicatie-intregratie die het Business Process hard codeert in de systeemarchitectuur



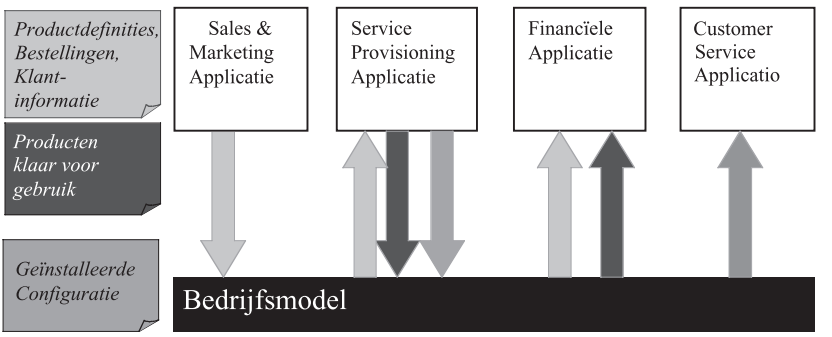
applicatie. Bij integratie door middel van een bedrijfslaag zoals afgebeeld in Figuur 14, gebeurt het doorgeven van informatie steeds via de bedrijfslaag. Op deze manier wint de implementatie aan flexibiliteit: het basisbedrijfsproces kan nu veel gemakkelijker aangepast worden. Bovendien is het in deze architectuur ook veel gemakkelijker een softwarepakket te vervangen: de basisinformatie zit steeds in de bedrijfslaag en gaat dus niet verloren.

V. VERDER ONDERZOEK

Het gebrek aan modularisatierichtlijnen in de huidige objecttgerichte systeemontwikkelingsmethoden wordt in MERODE opgevangen door een expliciet onderscheid te maken tussen het bedrijfsmodel en het functionaliteitsmodel. Doordat het bedrijfsmodel zich concentreert op de reële-wereldaspecten van het informatiesysteem en vrij is van technische beschouwingen, is het een beter instrument voor de communicatie met de bedrijfsmensen dan een klassieke systeemspecificatie. Bovendien is het opstellen van een bedrijfsmodel nuttig voor iedere organisatie omdat het inzicht geeft in de regels die gehanteerd worden binnen het bedrijf. Modelgedreven systeemontwikkeling laat ook toe systemen te ontwerpen volgens een gelaagde architectuur waardoor de onderhoudbaarheid en aanpasbaarheid sterk verbetert. Indien het bedrijfsmodel als kern van het informatiesysteem wordt geïmplementeerd, winnen systemen aan transparantie omdat de bedrijfsregels te traceren zijn tot in de implementatie. Dit is een bijkomende troef voor flexibele systemen.

FIGUUR 14

Voorbeeld van integratie door middel van een bedrijfsmodel



Door ook rekening te houden met de werkorganisatorische aspecten en deze niet te vermengen met de andere aspecten van het informatiesysteem behoudt men de mogelijkheid bedrijfsprocessen te reorganiseren zonder het informatiesysteem te moeten aanpassen.

De gelaagde architectuur die hier werd voorgesteld kan vergeleken worden met de klassieke architectuur met drie niveaus. In dergelijke three-tier architectuur vindt men bijvoorbeeld een client-applicatie-tier die de user interface aspecten en een deel van de applicatie-logica bevat, een server applicatie-tier die de rest van de applicatielogica bevat en tenslotte een database-tier [3]. Jacobson [4] identificeert drie soorten objecten: entiteit objecten (die samen het bedrijfsmodel vormen), controle-objecten en user interface objecten. Het grootste verschil met dit soort architecturen is dat in de MERODE-architectuur de verschillende soorten controle-logica over verschillende lagen verspreid zitten al naargelang het aspect waarop de logica betrekking heeft. Bedrijfsregels vindt men terug in de bedrijfslaag, applicatielogica zit in de functionaliteits- en user-interface-laag, business process aspecten zitten in de business-processlaag en de workflow service-laag bevat de logica die samenhangt met het gebruik van een workflow systeem.

De objectgerichte systeemontwikkelingsmethoden OO-SSADM [7] en Syntropy [1] erkennen net zoals MERODE het belang van bedrijfsmodellering. Het is interessant om te noteren dat deze methoden een aantal gemeenschappelijke kenmerken hebben met MERODE: ze zijn beïnvloed door de ideeën van JSD [5] en erkennen (dus) ook gebeurtenissen als fundamenteel modelleringsconcept. Ook Catalysis [2] kent het begrip “Action” dat op meer dan één object betrekking kan hebben. Eén van de toekomstige onderzoeksprojecten heeft betrekking op het uitbreiden van het concept “event” tot een algemeen multidirectioneel interactiemechanisme. Door het aanroepen van een event kan een softwarecomponent diensten vragen aan meerdere andere componenten tegelijkertijd en staat de event in voor het sturen en afhandelen van de choreografie van het geheel aan interacties. De software-componenten kunnen hierbij zowel eenvoudige objecten zijn als complexe, eventueel zelfs gedistribueerde componenten.

Het informele van veel huidige methoden legt een hypotheek op de kwaliteit van systeemontwikkeling. De complexiteit van specificaties is snel te groot om ze manueel en feilloos op consistentie te controleren. Formele methoden en goed gedefinieerde technieken kunnen het aantal fouten drastisch terugdringen. Zij zouden bovendien moeten toelaten om intelligente Computer-Aided Software Engineering

(CASE)-tools te bouwen die de consistentie van specificaties op een geautomatiseerde manier kunnen nagaan. In het verleden werd reeds onderzoek verricht naar het verifiëren van volgordebeperkingen op gebeurtenissen: deze kunnen contradicties bevatten die kunnen leiden tot deadlocks. Het geheel aan bedrijfsregels is echter veel uitgebreider dan het opleggen van volgordebeperkingen aan gebeurtenissen alleen. Toekomstig onderzoek moet meer inzicht verschaffen in de verschillende vormen van bedrijfsregels, en hoe en in welke laag men deze regels het best implementeert. Op dit ogenblik bestaat een eerste prototype van een CASE-tool die de huidige consistentie-controles van MERODE ondersteunt. De komende jaren zal deze tool verder uitgebouwd worden en zullen nieuwe inzichten omtrent het beheren en analyseren van bedrijfsregels er mee in geïntegreerd worden. Uiteindelijk zou men moeten kunnen komen tot een intelligent hulpmiddel dat de analist op een constructieve manier begeleidt bij het opstellen van een foutloos en kwalitatief hoogstaand bedrijfsmodel dat tevens een basisplan is voor toekomstige informatiesystemen.

NOTEN

1. MERODE is een acroniem voor Model-driven Existence dependency Relationship Object-oriented DEvelopment.
2. Ook het concept van generalisatie/specialisatie moet op analyseniveau anders ingevuld worden dan op programmatieniveau. Dit wordt o.m. besproken in [12].
3. De gegeven kritiek slaat enkel op het gebruik van message passing voor bedrijfsmodellering, niet op het gebruik voor programmeren.
4. Noteer dat vóór de opkomst van de objectgerichte technologie de organisatie-aspecten dikwijls ook niet opgenomen waren in het systeemontwikkelingstraject.

REFERENTIES

Cook, S., Daniels, J., 1994, Designing Object Systems: Object-Oriented Modeling with Syntropy, (Prentice Hall).

D'Souza, D.F., Wills, A. C. Wills, 1999, Objects, Components and Frameworks with UML, The Catalysis Approach, (Addison-Wesley), 785.

Fowler, M., 1997, Analysis Patterns, Reusable Object Models, (Addison Wesley Longman), 357.

Jacobson, I., Christerson, M., Jonsson P. et al., 1997, Object-Oriented Software Engineering. A Use Case Driven Approach, (Addison Wesley), Rev. 4th pr.

Jackson, M.A., 1983, System Development, (Prentice Hall, Englewood Cliffs, N.J.).

Rational Software Corporation, the Unified Modeling Language, http://www.rational.com/.

Robinson, K., Berrisford, G., 1994, Object-Oriented SSADM, (Prentice Hall).

Snoeck, M., Poels, G., 2000, Improving the Reuse Possibilities of the Behavioral Aspects of Object-Oriented Domain Models, in: Proceedings of the 19th International Conference on Conceptual Modeling (ER2000), (Salt Lake City), (Springer Verlag).

Snoeck, M., Poelmans S., Dedene G., 2000, A Layered Software Specification Architecture, in: Proceedings of the 19th International Conference on Conceptual Modeling (ER2000), (Salt Lake City), (Springer Verlag).

Snoeck M., Dedene G., 1998, Existence Dependency: the Key to Semantic Integrity Between Structural and Behavioral Aspects of Object Types, *IEEE Transactions on Software Engineering,* 24/24, 233-251.

Snoeck M., Bestaansafhankelijkheid: conceptueel modelleren "volgens contract", *Beleidsinformaticatijdschrift*, 96/4, (Contactgroep Beleidsinformatici, Leuven).

Snoeck M., Dedene G., Verhelst M; Depuydt A.M., 1999, Object-Oriented Enterprise Modeling with MERODE, (Leuven University Press, Leuven).

Wolber D., 1997, Reviving Functional Decomposition in Object-Oriented Design, *JOOP* 1997/10, 31-38.